

# TELEMETRY MONITORING AND DISPLAY USING LABVIEW\*

by

George Wells, Member of Technical Staff  
Edmund C. Baroth, Ph.D. Manager of the Measurement Technology Center,  
Jet Propulsion Laboratory, California Institute of Technology

\* The research described in this paper was carried out by the Jet propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

## Abstract

The **Measurement** Technology Center of the Instrumentation Section configures automated data acquisition systems to meet the diverse needs of the Jet Propulsion Laboratory's (**JPL's**) experimental research community. These systems are based on personal computers or workstations (Apple, IBM/Compatible, Hewlett-Packard and Sun Microsystems) and often **include** integrated data analysis, visualization and experiment control functions in addition to data acquisition capabilities. These integrated systems may **include** sensors, signal conditioning, data acquisition interface cards, software, and a user interface. Graphical programming is **used** to simplify configuration of such systems.

Employment of a graphical programming language is the most important factor in enabling the implementation of data acquisition, analysis, display and visualization systems at low cost. Other important factors are the use of commercial software packages and off-the-shelf data acquisition hardware where possible. Understanding the experimenter's needs is also critical. An interactive approach to user interface construction and training of the operators is also important.

An example of a telemetry monitoring and display application using two Macintosh computers and a graphical programming language (National Instruments' LabVIEW 2) will be discussed. One computer acted as the **telemetry** source and the other as the analyzer. The telemetry stream was emulated using interface boards in the computers. A schematic of the system and user interface panel will be presented. The computer programs will also be discussed as to ease of creation. The purpose of this paper is to show how using graphical-based programming software can be used for advanced data analysis like telemetry monitoring and display, as well as for emulation of a telemetry stream.

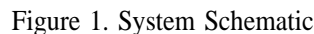
This application was created as a result of a 'competition' between a graphical programming language team and a text-based ('C') programming **team** to verify the advantages of using a graphical programming language approach. With approximately eight weeks of funding over a period of three months, the text-based programming team accomplished about 10% of the basic requirements, while the **Macintosh/LabVIEW** team accomplished about 150%, having gone beyond the original requirements to simulate a telemetry stream and provide utility programs. This application verified that using graphical programming can significantly reduce software development time. As a result of this initial effort, additional follow-on work was awarded to the graphical programming team.

## Introduction

As part of the Instrumentation Section, the Measurement Technology Center (**MTC**) evaluates data acquisition hardware and software products for inclusion into the Instrument Loan Pool for JPL experimenters. As such, it acts as a focus for off-the-shelf data acquisition, analysis and display hardware and software, including graphical programming software.

Using graphical **programming** software and off-the-shelf hardware, the **MTC** configures turn-key data acquisition systems customized to users' requirements. These systems include transducers, data storage capability, and a user **interface**. They may also include integrated data analysis, visualization and experiment-control functions in addition to data acquisition capabilities. The **MTC** supports systems based around IBM-compatible and Apple Macintosh personal computers, plus Hewlett-Packard and Sun Microsystems workstations. The advantages of graphical programming **languages** have been the key to configuring systems within a reasonable time and cost. Graphical programming has literally changed the way the Instrumentation Section does business. It has changed the way industry does business as well.<sup>3,4,5</sup>

The real potential of graphical programming, however, is the ability to go from conception to simulation of components, sub-systems and systems, to testing of actual hardware and control functions using a single software environment (on multiple platforms). Modules or icons that represent simulations of instruments, processes or algorithms can be easily replaced with the actual instruments or components when they become available. That potential was demonstrated in this application, as first the telemetry generator and analyzer were simulated in one computer, then were split to emulate the actual telemetry stream,



The telemetry data stream consists of a **two-line** serial interface (data and clock). Although it only needs to operate at 200 bits per second, it was tested at up to 5000 bits per second to measure the CPU margin.

# TELEMETRY MONITORING AND DISPLAY USING LABVIEW by George Wells & Edmund C. Baroth

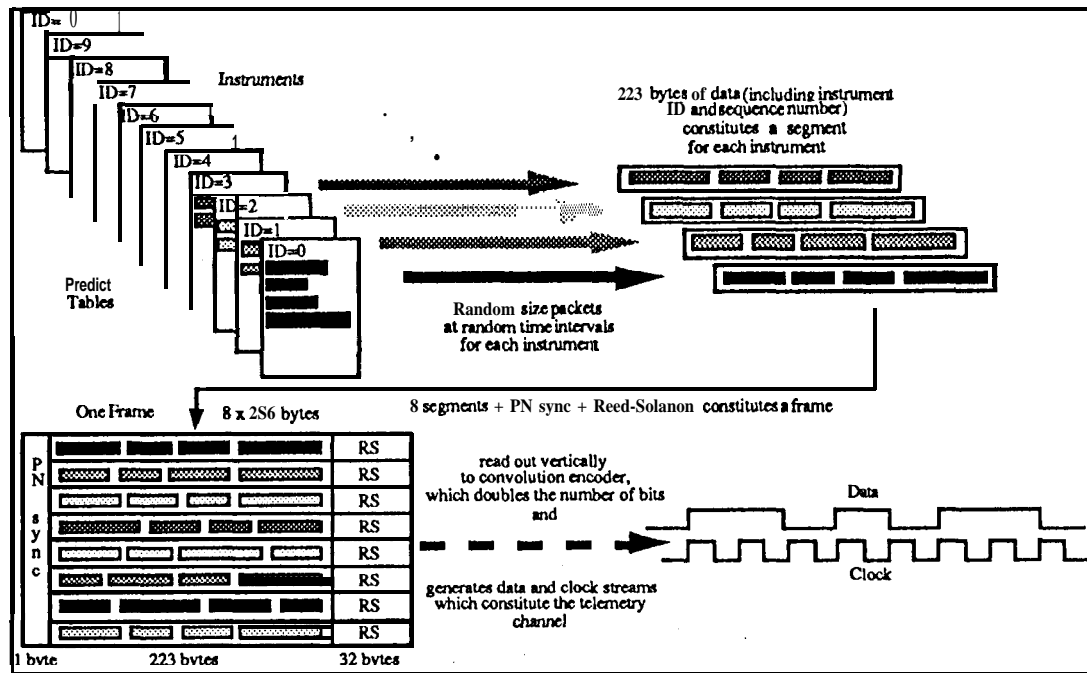


Figure 2. Telemetry Generator Sequence

## Telemetry Generator

There was no actual requirement for a telemetry generator, only an analyzer. The generator was created to test the analyzer. Figure 2 is the Telemetry Generator Sequence. The source of the data is pre-determined 'Predict Tables' containing random bytes. For each instrument there is a separate table. These tables are stored in both the generator and the analyzer computers. The overall approach is that each instrument sends packets of up to 220 bytes at random intervals of time and eventually these packets are picked up by the analyzer and verified in its Predict Tables. Packets that are received out of sequence or are not present in the Predict Tables will register as errors. It is a verification of the transmitting and receiving process, not the data itself.

The entire telemetry scheme can be thought of as having an independent channel for each instrument. Packets from an instrument are not actually sent until enough are received to fill a segment of 223 bytes (including some overhead). A Reed-Solomon error-correcting code of 32 bytes is appended to each segment and when eight segments are assembled (not necessarily from the same instrument), an eight-byte PN Sync word is attached. The bytes are sent starting with the PN Sync word and continue through the first byte of each segment followed by successive bytes of each segment. Before actually being sent over the telemetry channel, the stream is run through a convolution algorithm which provides additional error-correcting capabilities but doubles the number of bits.

The implementation of the various algorithms (e.g., Reed-Solomon and Convolution Coders) has traditionally been done in hardware using shift-registers, ex-OR gates and counters. The close analogy of LabVIEW's icons to actual circuitry enables easy and straight forward implementation of otherwise complex coding.

Figure 3 is the Front Panel for the telemetry generator. The range of packet sizes and time intervals can be specified for each instrument as well as the data rate. As each packet is generated, it is displayed on the strip chart as a dot. For example, Instrument ID= 0 attempts to output a packet of between 2 and 4 bytes every 1/6 of a second, while ID = 1 attempts to output packets containing between 20 and 132 bytes every 1 to 2 seconds. These rates are limited by the Bit Rate set on the front panel. Two types of errors can also be created to test the diagnostic capabilities of the analyzer.

TELEMETRY MONITORING AND DISPLAY USING LABVIEW  
by George Wells& Edmund C. Baroth

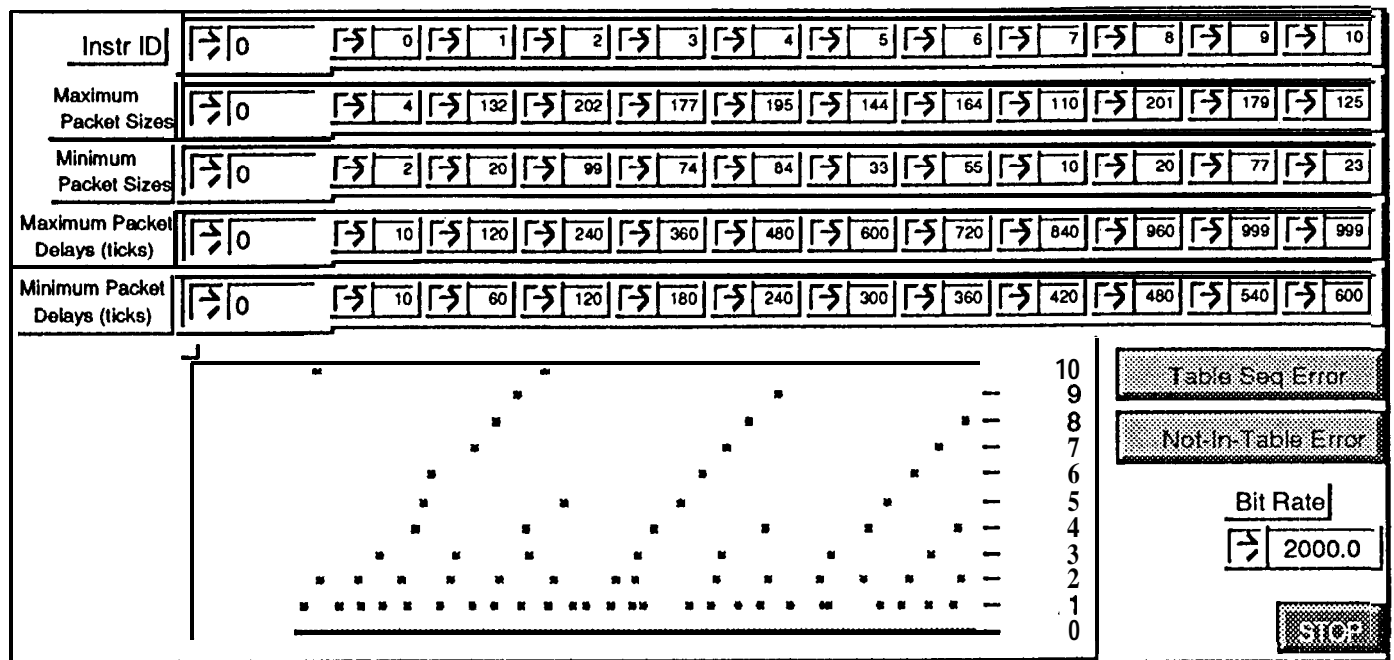


Figure 3. Telemetry Generator Front Panel

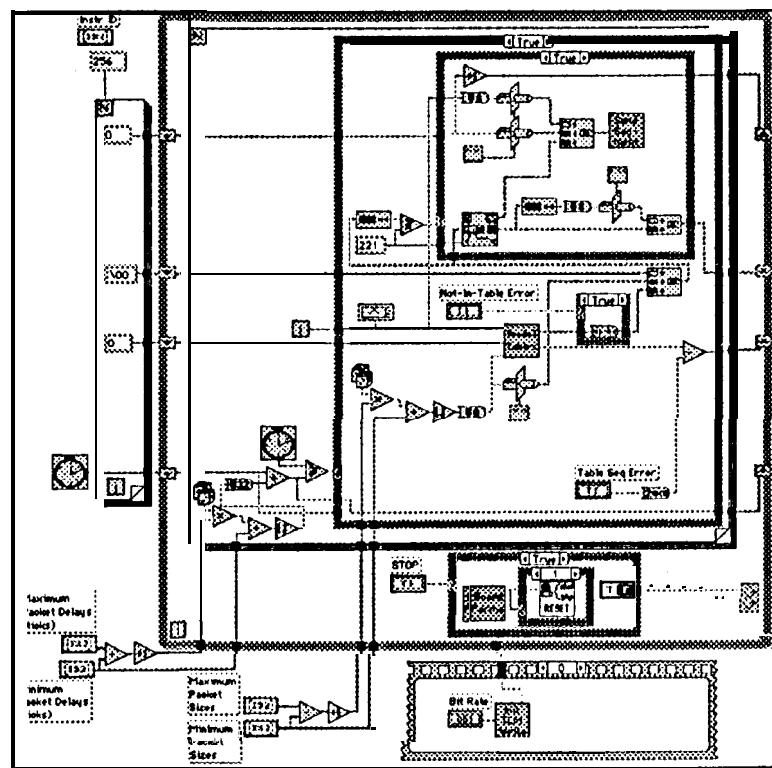


Figure 4. Telemetry Generator Diagram

Figure 4 is the diagram for the telemetry generator. It is responsible for temporarily storing the packets from each instrument until enough are available for a segment, at which time the 'Send Segment' VI (Virtual Instrument) is executed. This VI eventually calls the 'Calc Reed Solomon' VI shown in Figures 5 & 6 which implements all of the circuitry shown in Figure 7, but in a more general sense, in that the shift registers are put into a loop. The coefficients and RS Table are **permanently** stored as default values on the front panel. Notice how compact the block diagram is compared to the schematic "which it emulates.

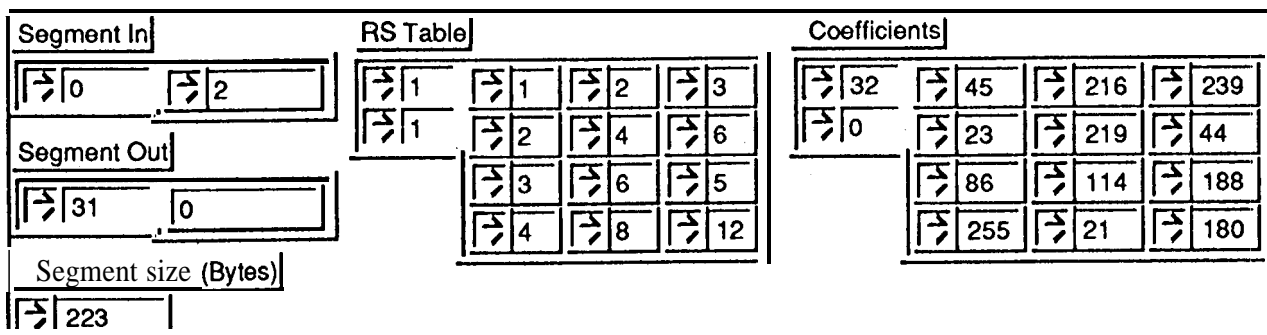


Figure 5. Calculate Reed-Solomon Front Panel

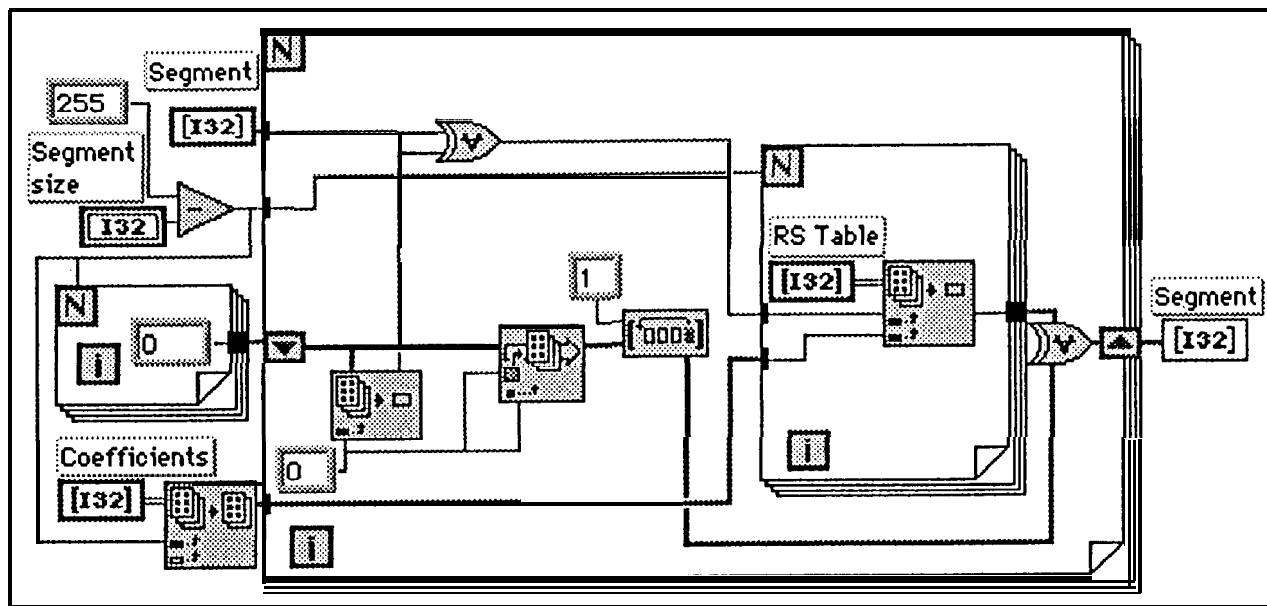


Figure 6. Calculate Reed-Solomon Diagram

After a complete telemetry frame is **assembled**, the bits are passed through a convolution coder which doubles the number of bits. The diagram in Figure 8 implements the hardware **circuitry** shown in Figure 9. The bits are eventually doubled again (so that each bit is present for two **clock** times) and used to control two voltage levels (zero and five volts) on one channel of the double-buffered D/A converter. The other channel generates an alternating bit **pattern** to form the clock. This completes the description of the Telemetry Generator.

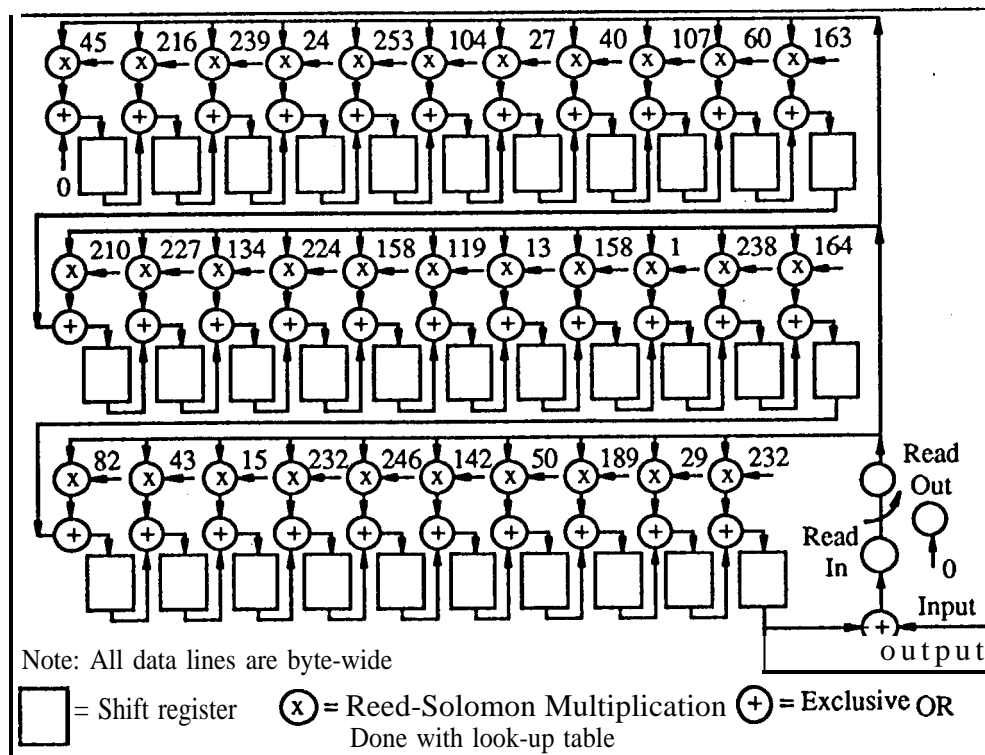


Figure 7. Reed-Solomon Circuitry

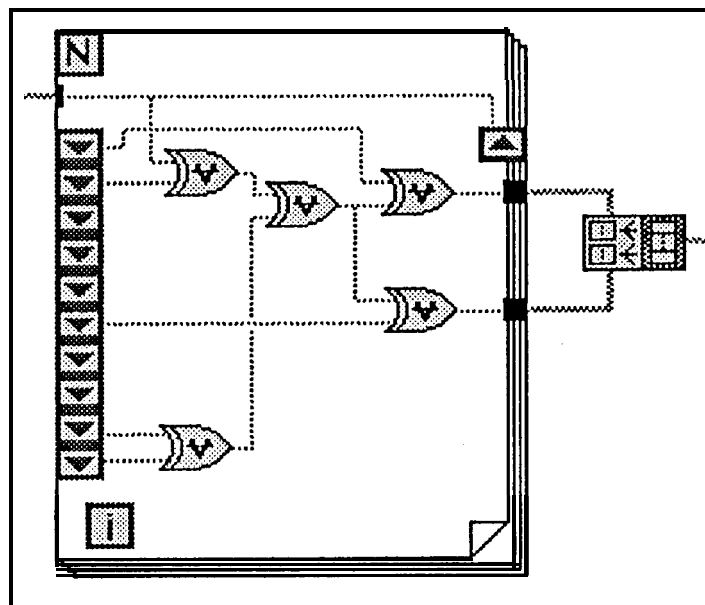


Figure 8. Diagram of Convolution Coder

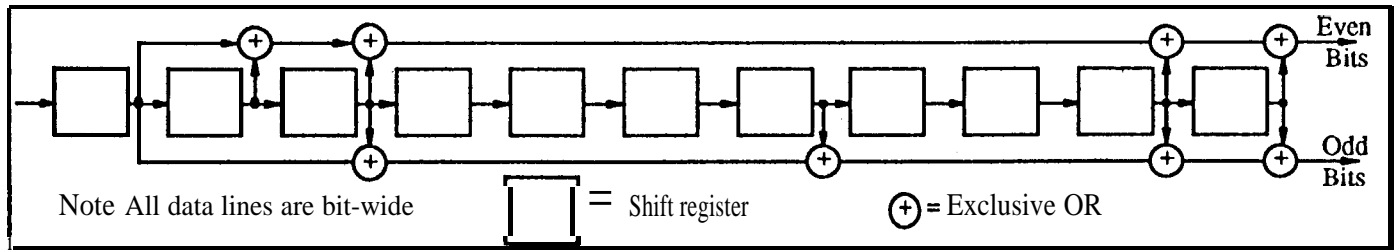


Figure 9. Convolutional Coder Circuitry

### Telemetry Analyzer

The Telemetry Analyzer uses the double-buffered analog input with samples taken on each falling edge of the clock. Each analog sample is immediately converted to a Boolean by a simple comparison test and passed through the convolution decoder shown in Figure 10 which implements the circuitry shown in Figure 11. Higher level VI's can request any number of bits from the convolutional decoder, but normally only half the bits are used. The bits come out in two separate arrays but only one of them is significant. The correct one is the one that contains the PN Sync word so the Get Sync VI shown in Figure 12 must search for the PN Sync word in both arrays. If it finds it in the second array, it reads and discards one more bit so-that the remainder of the frame is read from the first **array**.

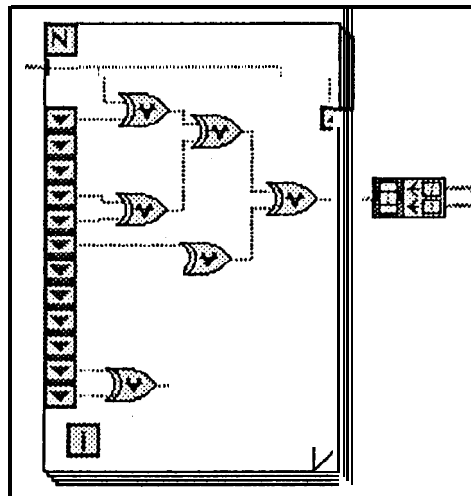


Figure 10. Diagram of Convolution Decoder

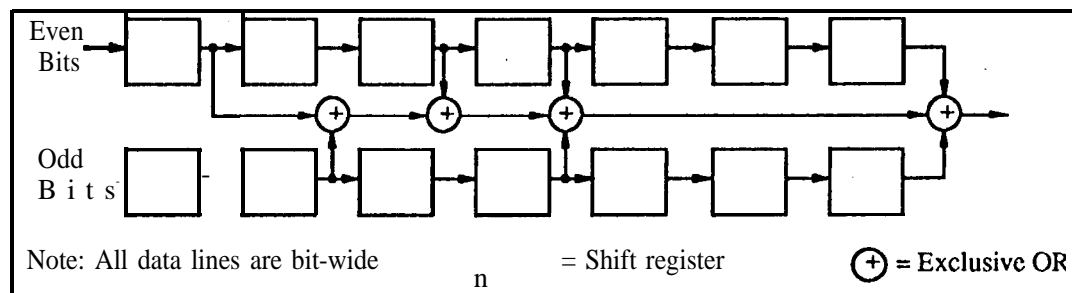


Figure 11. Convolutional Decoder Circuitry

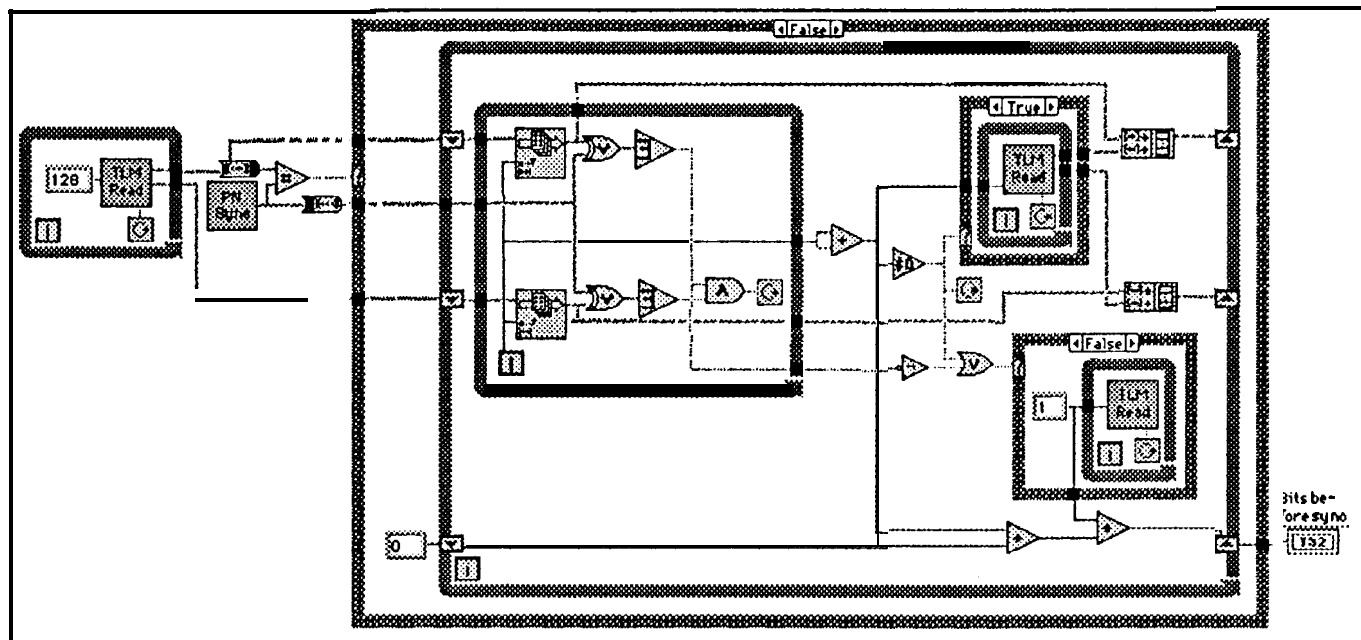


Figure 12. Diagram of Get Sync VI

The remainder of the Telemetry Analyzer basically performs the same functions that the Generator performs but in the opposite order. The same Reed Solomon calculations are done on each segment but instead of correcting errors they are simply flagged on the front panel (Figure 13). The header information from the packets is then stripped away and the data compared to the Predict Tables and appropriate error flags set. The VI has front panel logging turned on so that the status from each frame can be saved to disk. Each frame will have status information for eight segments including the Instrument ID, the current sequence number, the previous sequence number, error flags to indicate if the Reed-Solomon code is incorrect, if the sequence is not properly incremented, if the packet bytes were found in the Predict Table but not in the proper place, and if the packet bytes were not found in the Predict Table. The strip chart indicates which Instrument ID's each of the eight segments go with but it is not useful for data logging since only the last segment of each frame will be logged. Several other parameters relating to the frame itself are also indicated at the top of the front panel including the frame count, the number of bits found before the PN Sync code (which should always be zero after the first frame), the current time, the actual data rate, and finally the percent margin assuming a data rate of 2000 bits per second. All testing was done at ten times the targeted rate to save testing time.

Several utility VI's (Figure 14) are available to read the previously logged data, even while logging is in progress. Among these are a monitor to graph any of the parameters on the front panel against any of the others, e.g., to plot the bit rate against the frame count; a monitor to indicate Instrument ID utilization in the form of a bar graph; and a scrolling graph to display Instrument ID's as a function of real time,

## Conclusions

With approximately eight weeks of funding over a period of three months, the text-based programming team accomplished about 10% of the basic requirements, while the Macintosh/LabVIEW team accomplished about 150%, having gone beyond the original requirements to simulate a telemetry stream and provide utility programs. LabVIEW was shown to perform advanced data analysis tasks such as telemetry stream emulation and monitoring plus display. It also validated the graphical programming approach, as well as an interactive approach to user interface construction. This task succeeded in convincing people that graphical programming can significantly reduce software development time compared to text-based programming. It resulted in follow-on work for the Macintosh/LabVIEW team.

TELEMETRY MONITORING AND DISPLAY USING LABVIEW  
by George Wells& Edmund C.Baroth

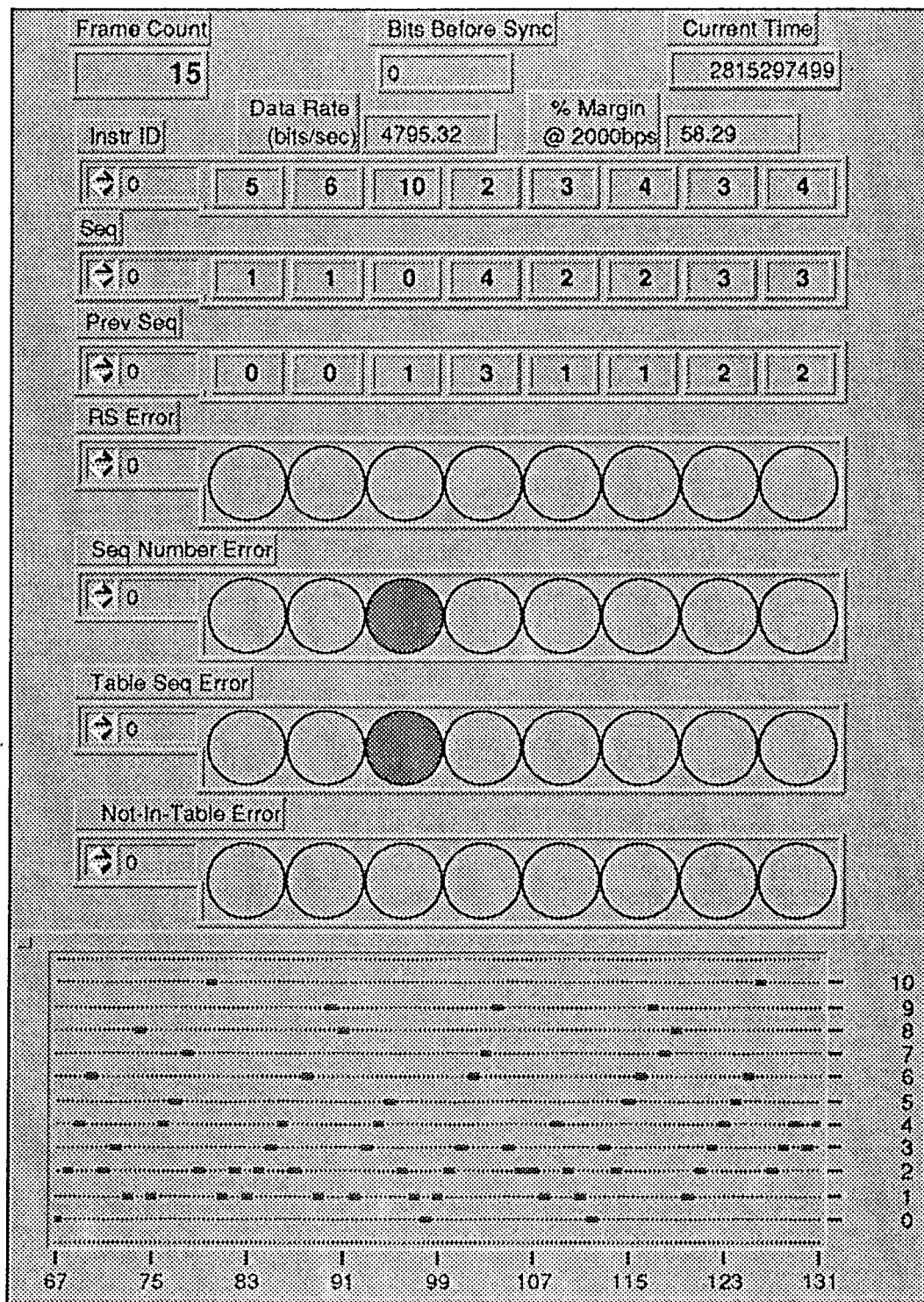


Figure 13. Telemetry Analyzer Front Panel

TELEMETRY Monitoring AND DISPLAY USING LABVIEW  
by George Wells& Edmund C. Baroth

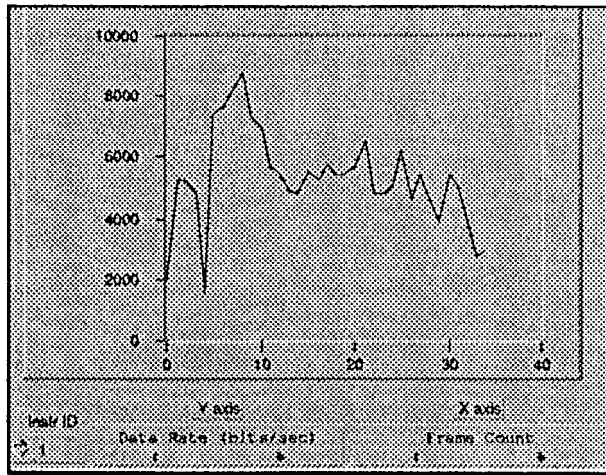


Figure 14a. Bit Rate vs. Frame Count

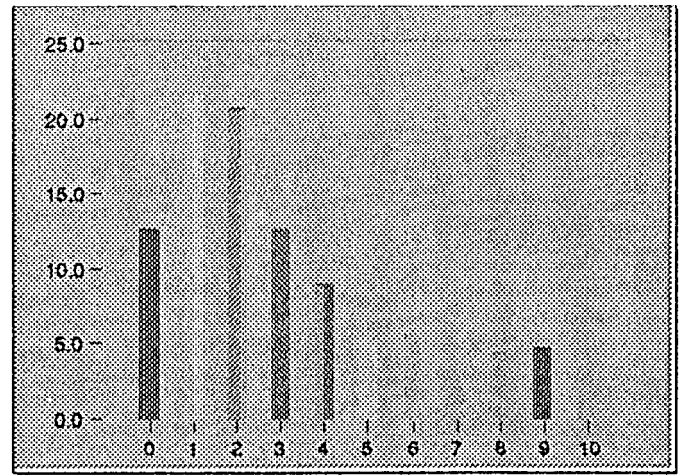


Figure 14b. Instrument ID Utilization

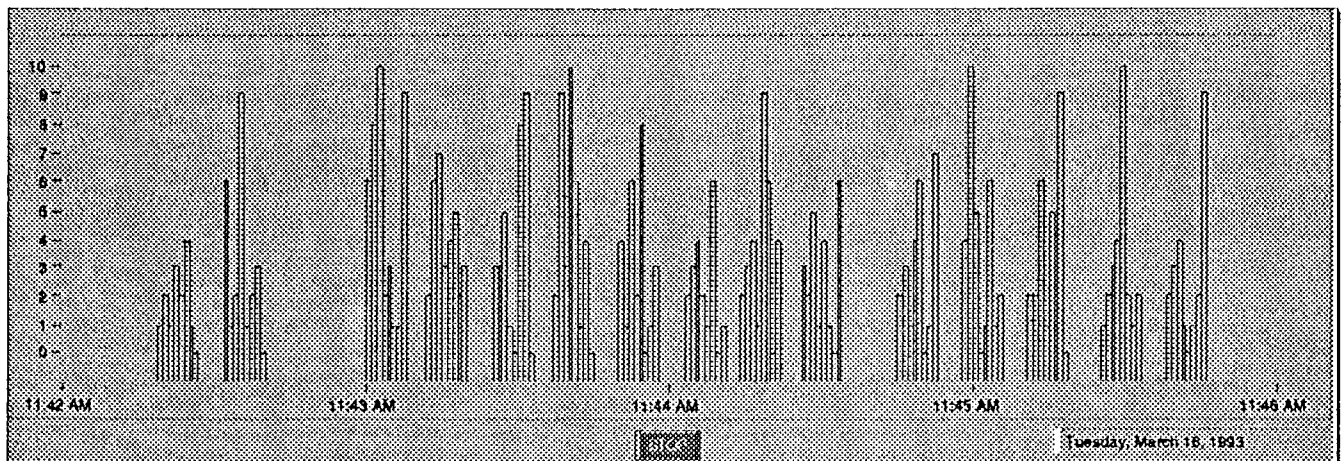


Figure 14c. Instrument ID vs. Real-Time

## References

1. *An Adaptive Structure Data acquisition System using a Graphical-Based Programming Language*, E. C. Baroth, D. J. Clark and R. W. Losey, Fourth AIAA/Air Force/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, Ohio, September 21-23, 1992.
2. *Acquisition, Analysis, Control, and Visualization of Data Using Personal Computers and a Graphical-Based Programming Language*, E. C. Baroth, D. J. Clark and R. W. Losey, Conference of American Society of Engineering Educators (ASEE), Toledo, Ohio, June 21-25, 1992.
3. *Automated RF Test System for Digital Cellular Telephones*, G. Kent, Proceedings from NEPCON West '93, Anaheim, CA, February 7-11, 1993, pp 1055-1064.
4. *Sequential File Creation for Automated Test Procedures*, J. R. Henderson, Proceedings from NEPCON West '93, Anaheim, CA, February 7-11, 1993, pp 1065-1077.
5. *Cutting Costs the Old Fashioned Way*, S. C. Jordan, Proceedings from NEPCON West '93, Anaheim, CA, February 7-11, 1993, pp 1921-1931,